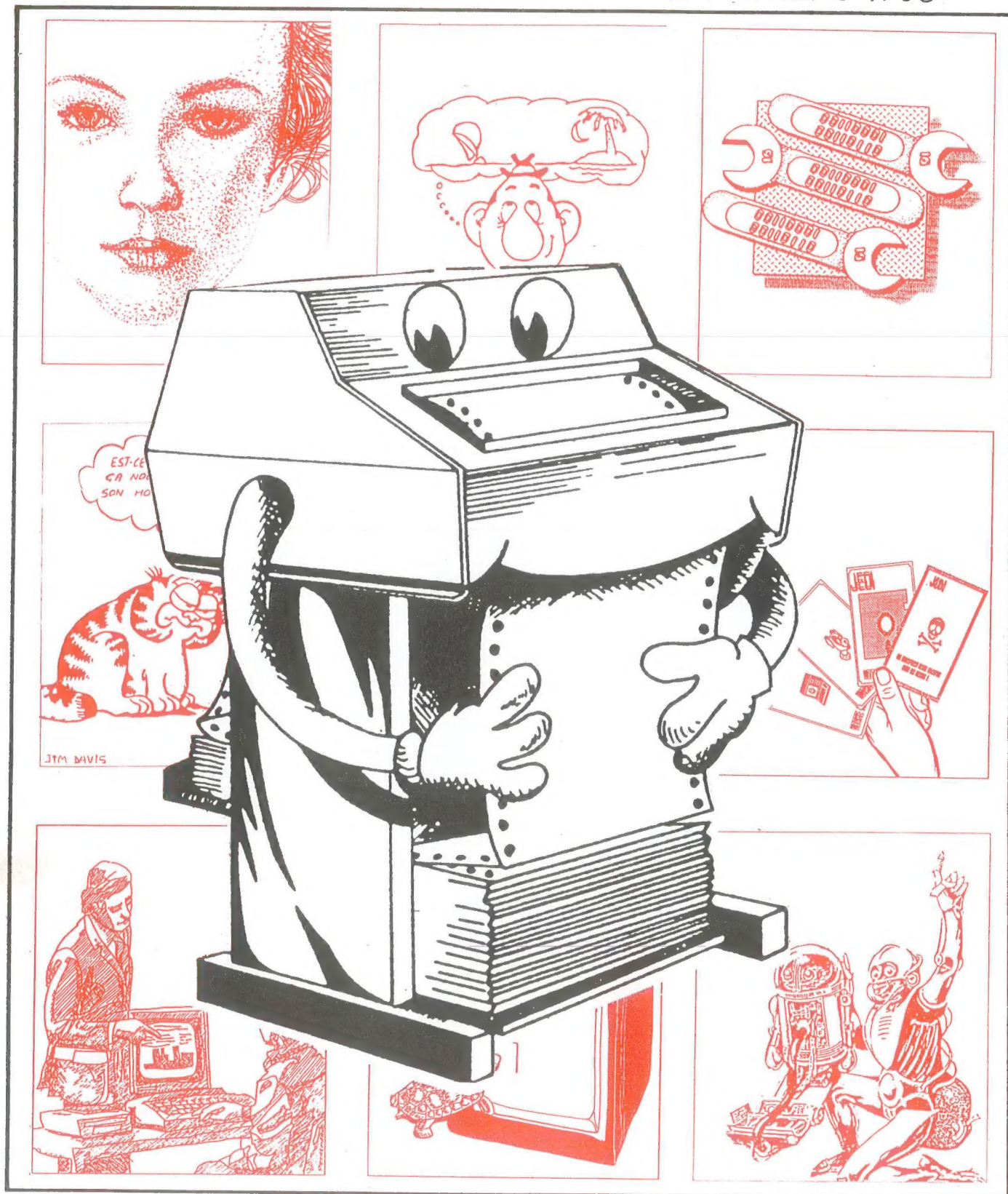


JEDI

31

Décembre 1986



EDITORIAL

De grandes nouveautés vous attendent pour cette nouvelle année dans l'univers de la micro-informatique:

- tout les micros seront sous MSDOS ou ils ne seront plus.
- les IBM se "Mac-intoshiseront" (GEM-like) et les Mac seront ouverts c'est à dire pourront exploiter les logiciels MSDOS.
- tout viendra de TAIWAN, SINGAPOUR, HONG-KONG bref, les claviers seront jaunes...
- ceux qui sortiront du standard de BIG-BLUE seront condamnés à l'ignorance (d'abord on siffle, puis on tire si on n'obtempère pas).

Ceci pour la note d'optimisme coté micro.

Une nouvelle maladie a fait son apparition: le SIDA mental (découvert par le directeur et éditorialiste du FIGARO MAGAZINE): seul remède, attendre que le sujet vieillisse ou s'abonne au FIGMAG (à bannir dorénavant de toutes les bonnes salles d'attentes).

Un nouveau langage a fait son apparition: le FORTH 83-Standard (la gestation a été longue); le FIGMAG a passé cette naissance sous silence. Peut-être sommes nous contaminés...

Une nouvelle façon de voir l'univers: le livre de Mr Hubert REEVES, l'heure de s'enivrer. Se dévore sans provoquer d'indigestion. Sa lecture devrait être conseillée à Mr PAUWELLS.

Une nouvelle année enfin, que l'on vous souhaite de tout coeur, pleine de réussite et la concrétisation de tous vos espoirs.

SOMMAIRE

APL	Gestion des tableaux	2
LOGO	Une gestion de fichiers	3
FORTH	Editeur pleine page pour AMSTRAD CPC	5
	Fonction horloge en temps réel pour IBM	7
	Structures de données PL/I en FORTH	9
	Fonctions graphiques pour AMSTRAD CPC	12
	Impression graphique multi-écran pour HRX	15
	Opérations en notation scientifique	16
	La gestion des fichiers en F83	18
	Sauvegarde des blocs F83 pour AMSTRAD PCW	18

Toute reproduction, adaptation, traduction partielle du contenu de ce magazine, sous toutes les formes est vivement encouragée, à l'exclusion de toute reproduction à des fins commerciales. Dans le cas de reproduction par photocopie, il est demandé de ne pas masquer les références inscrites en bas de page, et dans les autres cas, de citer l'ASSOCIATION JEDI. Pour tout renseignement, vous pouvez nous contacter en nous écrivant à l'adresse suivante:

ASSOCIATION JEDI 8, rue Poirier de Narçay 75014 PARIS

Tel: (1) 45.42.88.90 (de 10h à 18h)

Une des caractéristiques les plus intéressantes de l'APL, est sa capacité à traiter les variables multidimensionnelles: vecteurs, matrices, tableaux.

Nous n'allons pas faire ici un exposé complet sur le traitement de ces variables, mais simplement tenter d'en donner un aperçu au non-APListe.

Considérons le petit problème suivant: Pierre revient du marché avec 1 pomme, 2 poires et 3 oranges. De son côté, Paul rapporte 3 pommes, 1 poire et pas d'orange. Nous allons créer deux variables:

```
PIERRE ← 1 2 3
PAUL   ← 3 1 0
```

Chacune de ces variables comporte trois valeurs "chainées" ensemble. On leur donne le nom de "vecteurs" de longueur 3. Si on veut savoir ce qu'il y a dans la variable PIERRE, on frappe:

```
PIERRE
```

et la réponse est: 1 2 3

Si l'on veut savoir la longueur de la variable, on utilise la fonction ρ (rho) sous la forme monadique en frappant:

```
 $\rho$  PIERRE
```

La réponse est: 3

PIERRE est un vecteur de longueur 3. Si l'on veut savoir combien de fruits auront été rapportés, il suffira de frapper:

```
PIERRE + PAUL
```

La réponse sera: 4 3 3

C'est à dire 4 pommes, 3 poires et 3 oranges. Si une pomme coûte 1,5 Fr, une poire 1 Fr et une orange 2 Fr (Ndlr: bigre, quelle inflation!!!...), on peut connaître les sommes payées par PIERRE en frappant:

```
PIERRE * 1.5 1 2
```

résultat: 4.5 2 6

PIERRE aura acheté pour 1,5 Fr de pommes, pour 2 francs de poires et pour 6 francs d'oranges.

On aurait pu créer une variable:

```
PRIX ← 1.5 1 2
```

et on aurait obtenu le même résultat en frappant:

```
PIERRE * PRIX
```

On pourra même avoir la somme totale payée par PIERRE en utilisant la fonction "réduction" (\uparrow). Sans entrer ici dans des détails de syntaxe, sachons seulement que les symboles \uparrow placés devant un vecteur provoquent le calcul de la somme des éléments du vecteur:

```
 $\uparrow$  (PIERRE * PRIX)
```

donnera: 9.5

Soit la somme payée par PIERRE. Bien entendu, on peut en faire autant pour PAUL. On aurait pu également obtenir le même résultat en faisant le "produit interne" des vecteurs PIERRE et PRIX, qui s'écrit:

```
PIERRE +. * PRIX
```

Les symboles \uparrow provoquent les produits des éléments correspondants de chaque vecteur, puis la somme de ces produits. Mais n'allons pas trop loin pour l'instant.

Nous aurions pu rassembler les données relatives à PIERRE et à PAUL en une seule variable bi-dimensionnelle (tableau) dans laquelle les valeurs relatives à un même individu seraient sur une même ligne, et les valeurs relatives à un même fruit sur une même colonne.

Pour cela, il faut créer une variable à deux lignes et trois colonnes en utilisant la fonction ρ (rho) diadique en frappant:

```
TABLEAU ← 2 3  $\rho$  PIERRE, PAUL
```

La variable TABLEAU contient les valeurs suivantes:

```
1 2 3
3 1 0
```

La fonction ρ monadique permet de contrôler les dimensions de la variable:

```
 $\rho$  TABLEAU
```

résultat: 2 3

soit 2 lignes et 3 colonnes.

Le nombre total de fruits rapportés par l'un et l'autre se calcule à l'aide d'une réduction le long des lignes:

```
 $\uparrow$  TABLEAU
```

donne: 6 4

Soit 6 fruits pour PIERRE et 4 pour PAUL. Le nombre de chaque type de fruit est donné par une réduction le long des colonnes:

```
 $\uparrow$  TABLEAU
```

résultat: 4 3 3

soit 4 pommes, 3 poires et 3 oranges.

Un produit interne permet de connaître la somme globale payée par chacun:

```
TABLEAU +. * PRIX
```

donne: 9.5 5.5

soit 9,50 Fr pour PIERRE et 5,50 Fr pour PAUL. Une réduction de ce dernier résultat donne la somme totale payée par PIERRE et PAUL.

```
 $\uparrow$  TABLEAU +. * PRIX
```

donne: 15

suite page 8

LOGO

A>type logofich.txt

L'application LOGO que vous trouverez ci-jointe est un répertoire téléphonique grand format, 1000 noms possibles, pouvant servir de support d'étude des fichiers LOGO.

Configuration T07 + extension 16K + disquette simple face et densité (NdLR: ou T07-70 + disquette SFSD).

LES GRANDES LIGNES DE CETTE APPLICATION

Le fichier global des noms et numéros "TEL est rangé sur disquette en 11 tranches gérées par la liste "ARCH.

Seule sera présente en RAM une tranche de 100 noms (à l'aise).

L'échange se fait au moment où ayant tapé le nom (et validé), on cherchera des yeux le numéro de téléphone à inscrire.

Le système présente le ou les numéros des homonymes éventuels pour éviter les redondances.

Les réponses sont classées en liste dans l'ordre croissant des noms au fur et à mesure des entrées et les homonymes, le dernier en tete.

Des corrections par effacement sont possibles.

L'impression du répertoire par tranches.

La sauvegarde de la dernière correction où l'ajout se fait par "OFF avant extinction des feux.

QUELQUES OBSERVATIONS SUR LES ENCOMBREMENTS MEMOIRE

Un éditeur "OCCUP permet de définir des procédures de test d'occupation qui apportent des renseignements intéressants.

La liste "ELM permet de définir la taille du fichier. Telle que définie, elle générera des listes de listes de 10 caractères.

Le premier de chaque mot est un identificateur de ligne, les caractères qui suivent: la colonne:

? DONNE "TST MULTI [] 65 116

génère un encombrement

"ELM" réduit à un mot de 9 caractères:

10car x 50 = 500 demandent: 530 octets (il n'y a pas d'objections).

"ELM réduit à deux mots:

2 x 10 x 50 = 1000 demandent: 1340 octets (on est passé en deux dimensions?).

"ELM à 5 mots:

5 x 10 x 50 = 2500 demandent: 2830 octets (50 lignes, 5 colonnes, rien à dire).

LOGO ouvre une autre possibilité, celle de l'indirection que nous avons testée.

? DONNE "TST SIMPLE [] 65 116

génère une liste de mots définis par le premier de "ELM auquel on "DONNE le restant de cette liste. Le meme nombre de caractères n'est généré que dans le dernier cas, mais l'organisation et l'exploitation sont plus du tout le memes, l'encombrement non plus.

Pour 10 x 50 + 4 x 10 x 50 = 2500 caractères, l'encombrement passe à 3080 octets.

LES ENCOMBREMENTS LORS DE TRAITEMENT

Ils sont particulièrement intéressants à connaître. Deux méthodes sont possibles, à ma connaissance, et seulement deux:

- la récursivité non terminale permet de supprimer ou modifier un élément quelconque

EC *VRP-19

POUR INIT
DONNE *TEL [AAAAA A] [ZZZZZ Z]
FIN

POUR VERSE :ME
SI VIDE? :ME [STOP]
DONNE *TEL ORDRE INSR :TEL PH PREM PREM :ME PREM SP PREM :ME
VERSE SP :ME
FIN

POUR INSR :M :N
SI PK<SM PREM :N PREM PREM :M [REND MP :N :M]
REND INSR +LIF :M :N
FIN

EC *VRP-20

POUR HOMON :M :NOM
SI EGAL? :NOM PREM PREM :M [VISUL :AFI PREM :M :LON TAPE "QUI? SI EGAL?
D LISCAR [EC [] RENDS :M] [EC []]
SI PK<SM :NOM PREM PREM :M [TAPE [tel:] RENDS MP PH :NOM PREM LL :M]
REND HOMON +LIF :M :NOM
FIN

POUR ORDRE :M
SI ASCII PREM PREM :M < ASCII PREM DER :M [REND :M]
REND ORDRE +LIF :M
FIN

POUR PK<SM :P :S
SI VIDE? :P [REND VRAI]
SI VIDE? :S [REND FAUX]
SI EGAL? PREM :P PREM :S [REND PK<SM SP :P SP :S]
REND PLP? ASCII PREM :P ASCII PREM :S
FIN

POUR +LIF :L
REND MD PREM :L SP :L
FIN

POUR -LIF :L
REND MP DER :L SD :L
FIN

EC *VRP-21
DONNE *AFI [nom: tel:]

DONNE *LON [16 11]

POUR E
TAPE [Premier caractere ?]
DONNE *NOM PREM LL
SI NON NOM? *DSK [NARCH]
SI NON TAR? :DSK ASCII :NOM [OFF NARCH]
ENUM 1 SP :TEL
TAPE [NUM A EFFACER ?]
DONNE *NUM PREM LL
SI NOMBRE? :NUM [DONNE *TEL ORDRE ENLI :NUM + 1 :TEL]
FIN

POUR ENLI :N :L
SI EGAL? ITEM :N :TEL PREM :L [REND SP :L]
REND ENLI :N +LIF :L
FIN

POUR VISUL :A :M :L
SI VIDE? :A [STOP]
TAPE MOT PREM :A PREM :M
REPETE PREM :L - CO PREM :M [TAPE CAR 9]
VISUL SP :A SP :M SP :L
FIN

POUR CO :M
SI VIDE? :M [REND 0]
REND 1 + CO SP :M
FIN

DONNE "TST DEBAL :TST

provoque l'affichage de la place disponible après génération de "TST sans différentes formes. Il faut noter que l'occupation est considérable et indépendante de la forme, seule la longueur de liste importe (le nombre de lignes).

- la récursivité terminale avec une gestion circulaire permet le même travail, avec une occupation infime par rapport à la première méthode.

DONNE "TST CIRCUM : TST COMPTE :TST

4500 octets dans la 1ère méthode

430 octets dans la dernière méthode.

La première méthode permet une définition très claire, très élégante, mais à réserver pour les listes courtes.

LES PROCEDURES PRINCIPALES

Ajoute: POUR A

permet d'ajouter un nouveau nom avec gestion circulaire de la liste "TEL. Celle-ci a été ouverte avec des mots qui seront des identificateurs de début et de fin. Quelle que soit la déformation de cette liste, la procédure ORDRE rendra une liste avec ces identificateurs correctement rétablis.

Menu: POUR M

la procédure M est une commande vecteur qui écrit le menu en fonction du contenu de la liste "MENU, ceci permettant de valider un choix et un seul. Elle pourrait être éventuellement complétée par des effacements de procédures indésirables le cas échéant.

Nota: la limite en longueur de ce fichier n'est pas tellement l'occupation mémoire, mais le temps mis par le système pour installer une nouvelle tranche d'archive "TEL en mémoire. Ce temps n'est pas négligeable pour les grands fichiers.

POUR ENUM :P :L

SI VIDE? SD :L [STOP]

TAPE :P VISUL :AFI PREM :L :LON EC "

ENUM 1 + :P SP :L

FIN

EC "VRP-22

DONNE "ARCH [66 66 A-A/TEL] [65 67 B-B/TEL] [66 68 C-C/TEL] [67 69 D-D/TEL] [68 71 E-F/TEL] [70 72 G-G/TEL] [71 77 H-L/TEL] [76 78 M-M/TEL] [77 81 N-P/TEL] [80 84 Q-S/TEL] [83 91 T-Z/TEL]

POUR A

TAPE "noa? :

DONNE "NOM PREM LL

SI NOM NOM? "DSK [COLOR NARCH SUITE STOP]

SI TAR? :DSK ASCII :NOM [SUITE] [OFF NARCH SUITE]

FIN

POUR SUITE

DONNE "TEL ORDRE HOMON :TEL :NOM

FIN

POUR NARCH

DONNE "DSK SLCT :ARCH ASCII :NOM

RAMENE DER :DSK

FIN

POUR SLCT :LA :A

SI VIDE? :LA [RENDIS :L]

SI TAR? PREM :LA :A [RENDIS PREM :LA] [RENDIS SLCT SP :LA :A]

FIN

POUR TAR? :L :A

RENDIS ET PREM :L < :A PREM SP :L > :A

FIN

POUR OFF

DETRUIS DER :DSK

SAUVE DER :DSK ["MEMO]

FIN

POUR COLOR

TAPE MOT CAR 27 CAR 97

TAPE MOT CAR 27 MOT CAR 32 CAR 64

TAPE MOT CAR 27 MOT CAR 32 CAR 83

FIN

EC "VRP-23

POUR I

EC "IMPRESSION

SI NOM? "TEL [EFN "TEL]

SELAR :ARCH

SORTIE 2

EC "ELEM SP :TEL

SORTIE 1

FIN

POUR SELAR :L

SI VIDE? :L [STOP]

TAPE PH "de PH CAR 1 + PREM PREM :L PH "a PH CAR PREM SP PREM :L - 1 [QUI ?]

SI EGAL? "O LISCAR [RAMENE DER PREM :L STOP] [EC "]

SELAR SP :L

FIN

POUR ELEM :L

SI VIDE? SD :L [STOP]

VISUL :AFI PREM :L :LON EC "

ELEM SP :L

FIN

POUR M

COLOR

EXEC :MENU

FIN

POUR :TEL

DONNE "MENU [EC ["A'ajoute] EC ["E'dite,efface] EC ["OFF'sauve disque]]

SAUVE "TEL [HOMON ORDRE PKSM +LIF "AFI "LON E ENLI VISUL CO ENUM A SU

ITE NARCH SLCT TAR? OFF COLOR "MENU "ARCH M]

FIN

POUR :TEL

DONNE "MENU [EC ["I'prime]]

SAUVE "TEL ["AFI "LON VISUL CO COLOR "MENU "ARCH M I SELAR ELEM]

FIN

EC "OCCUP

DONNE "ELM [aaaaaaaaa bbbbbbbbbb ccccccccc dddddddd eeeeeeee]

POUR ALIGNE :L :CD :ID

SI VIDE? :ID [RENDIS :L]

RENDIS MP MOT CAR :CD PREM :ID ALIGNE :L :CD SP :ID :L

FIN

POUR LIDOLI :P :CD :ID

DONNE "P MOT CAR :CD PREM :ID

DONNE :P ALIGNE :L :CD SP :ID

RENDIS :P

FIN

POUR SIMPLE :L :CD :CF

SI EGAL? :CD :CF [RENDIS :L]

RENDIS MP LIDOLI " :CD :ELM SIMPLE :L 1 + :CD :CF :L

FIN

POUR MULTI :L :CD :CF

SI EGAL? :CD :CF [RENDIS :L]

RENDIS MP ALIGNE :L :CD :ELM MULTI :L 1 + :CD :CF :L

FIN

POUR DEBAL :L

SI VIDE? :L [EC PH "DISPONIBLE: PLACE RENDIS :L]

RENDIS MP PREM :L DEBAL SP :L

FIN

POUR +LIF :L

RENDIS MD PREM :L SP :L

FIN

POUR CIRCUM :L :N

SI EGAL? 0 :N [RENDIS :L]

RENDIS +LIF :L 1 - :N

FIN

FIG-Forth pour AMSTRAD

SCR # 36

```

0 ;S      Pour les nordus du Forth-Amstrad:
1
2 voici un Editeur Pleine-Page qui utilise les commandes de
3 l'editeur du Fig. Il est bien plus rapide et confortable a
4 utiliser que celui paru dans Jedi 25 (pardon a R.Jeannin, mais
5 merci pour sa copie d'ecran avec l'utilitaire de calculs en LM).
6 N'utilisez que les commandes citees apres vous etre assures que
7 les codes Ascii de votre clavier concordent avec ceux utilises
8 ici ( sinon servez-vous du SETUP.COM ou modifiez-les ).
9 En cas de bugs residuels ou pour toute autre information, voici
10 ou se joindre:
11          BARRAUD Alain      (membre Jedi)
12          12 rue des sapins
13          77210 AVON
14
15          tel: (16.1) 60.72.25.43 (19H-22H ou Week-end)
    
```

SCR # 37

```

0 ( Commandes de l'Editeur Pleine Page      ABSEP86 )
1 ;S
2 FLECHES-CURSEUR      deplacement dans l'ecran vers
3                        - haut
4                        - bas
5                        - gauche
6                        - droite
7
8 <CTRL> FLECHE-HAUT      amene le curseur en debut de ligne 0
9 <CTRL> FLECHE-BAS      amene le curseur en debut de ligne 15
10 <CTRL> FLECHE-GAUCHE   amene le curseur en debut de ligne
11 <CTRL> FLECHE-DROITE   amene le curseur en fin de ligne
12
13 <SHIFT> FLECHE-HAUT    liste l'ecran n-1
14 <SHIFT> FLECHE-BAS     liste l'ecran n+1
15
    
```

SCR # 38

```

0 ( Commandes de l'Editeur Pleine Page      ABSEP86 )
1 ;S
2 <CTRL> X      remplit l'ecran avec le caractere 'espace'.
3 <CTRL> E      efface la ligne ou se trouve le curseur.
4 <CTRL> D      ote la ligne ou se trouve le curseur,
5                celles du dessous remontent.
6 <CTRL> S      insere une ligne vierge, celles du dessous
7                descendent, la 15eme est perdue.
8 <CTRL> H      memorise dans PAD la ligne courante.
9 <CTRL> R      recopie en ligne courante la ligne memorisee
10                par <ctrl> H ( si elle est toujours dans PAD ).
11 <CTRL> I      insere en ligne courante la ligne memorisee
12                dans PAD, la 15eme est perdue ( meme remarque ).
13 <CTRL> <TAB>  tabulation de 8 .NE PAS UTILISER <TAB> SEUL !!
14                a cause de son code ASCII identique a <CTRL> I.
15 <CTRL> Q      retour en Forth, a force d'y penser.
    
```

SCR # 39

```

0 ( Commandes de l'Editeur Pleine Page      ABSEP86 )
1 ;S
2 <DEL>          efface le caractere sous le curseur, les
3                suivants se decalent vers la gauche, un
4                espace est ajoute en fin de ligne.
5
6 <COPY>         insere un espace a l'endroit du curseur,
7                le dernier caractere de la ligne est perdu.
8
9 <ENTER>        fait passer en debut de ligne suivante.
10
11 A part ces commandes, le texte s'inscrit la ou se trouve le
12 curseur.
13 Ne pas oublier de faire FLUSH apres avoir quitte l'edition.
    
```

SCR # 40

```

0 ( Editeur pleine page      ABSEP86 )
1
2 EDITOR DEFINITIONS DECIMAL
3
4 : BEEP          ( --- : fait 'bip' )
5   7 EMIT ;
6
7 : GAUCHE        ( --- : decremente R# de 1 )
8   R# @ 0 = IF BEEP
9     ELSE -1 R# +!
10    THEN ;
11
12 : DROITE        ( --- : incremente R# de 1 )
13   R# @ 1023 = IF BEEP
14     ELSE 1 R# +!
15    THEN ;
16
17 -->
    
```

SCR # 41

```

0 ( Editeur pleine page suite      ABSEP86 )
1 : HAUT          ( --- : decremente R# de 64 )
2   R# @ 64 < IF BEEP
3     ELSE -64 R# +!
4     THEN ;
5 : BAS           ( --- : incremente R# de 64 )
6   R# @ 959 > IF BEEP
7     ELSE 64 R# +!
8     THEN ;
9 : DEB-L15       ( --- : R# indique le debut de ligne 15 )
10  960 R# ! ;
11 : DEB-L         ( --- : R# indique le debut de ligne )
12  #LEAD MINUS R# +! DROP ;
13 : FIN-L         ( --- : R# indique la fin de ligne )
14  #LAG 1 - R# +! DROP ;
15
16 -->
    
```

SCR # 42

```

0 ( Editeur pleine page suite      ABSEP86 )
1 : <ENTER>       ( --- : R# indique le debut de ligne dessous )
2   R# @ 959 > IF DEB-L15 BEEP
3     ELSE #LAG R# +! DROP
4     THEN ;
5
6 : CURSEUR       ( --- : affiche le curseur )
7   #LOCATE 1+ SWAP 1+ LOCATE ;
8
9 : LIGNE         ( --- ligne )
10  #LOCATE SWAP DROP ;
11
12 : .LIGNE        ( ligne --- : affiche la ligne a sa place )
13  DUP 1+ 1 LOCATE
14  SCR @ .LINE ;
15
16 -->
    
```

SCR # 43

```

0 ( Editeur pleine page suite      ABSEP86 )
1 : .ECRAN        ( --- : liste l'ecran )
2   16 0 DO FORTH I .LIGNE
3   LOOP ;
4
5 : ERA-LIGNE     ( ligne --- : efface la ligne courante de l'ecran )
6   1+ 1 LOCATE 18 EMIT ;
7
8 : CTRL_E        ( --- : efface la ligne en buffer et ecran )
9   LIGNE DUP E ERA-LIGNE ;
10
11 : CTRL_D        ( --- : ote la ligne courante buffer et ecran )
12   LIGNE DUP D 1+ 1 LOCATE 20 EMIT .ECRAN ;
13
14
15
    
```

```

SCR # 44
0 ( Editeur pleine page suite ABSEP86 )
1
2 : CTRL_X ( --- : efface l'ecran buffer et ecran )
3   SCR @ CLEAR CLS ;
4
5 : EN-TETE 0 #WINDOW 1 20 LOCATE
6   ." Editeur Pleine Page : SCR # " SCR @ 3 .R ;
7
8 : SCR-1 ( --- : montre l'ecran precedent )
9   SCR @ 1 = IF BEEP
10  ELSE -1 SCR +! EN-TETE 1 #WINDOW CLS TOP .ECRAN
11  THEN ;
12 : SCR+1 ( --- : montre l'ecran suivant )
13   SCR @ 179 = IF BEEP
14   ELSE 1 SCR +! EN-TETE 1 #WINDOW CLS TOP .ECRAN
15   THEN ; -->

```

```

SCR # 45
0 ( Editeur pleine page suite ABSEP86 )
1
2 : INS ( --- : insere un espace sous le curseur )
3   #LAG PAD SWAP CMOVE
4   BL #LAG DROP C!
5   PAD #LAG 1 - >R 1+ >R CMOVE
6   CURSEUR 18 EMIT #LAG TYPE UPDATE ;
7
8
9 : DEL ( --- : ote le caractere sous le curseur )
10  PAD C/L 1+ BL FILL
11  #LAG PAD SWAP CMOVE
12  PAD 1+ #LAG CMOVE
13  CURSEUR 18 EMIT #LAG TYPE UPDATE ;
14
15 -->

```

```

SCR # 46
0 ( Editeur pleine page suite ABSEP86 )
1 : CTRL_TAB R@ @ 1016 < IF 0 R@ +! ( --- : tabule de 0 )
2   ELSE BEEP THEN ;
3
4 : ECRIT? ( c --- : ecrit un caractere imprimable )
5   DUP 31 < SWAP DUP 127 > ROT OR
6   IF DROP
7   ELSE DUP #LEAD + C! EMIT
8   THEN DROITE UPDATE ;
9
10 : CTRL_H ( memorise la ligne courante dans PAD )
11   LIGNE H ;
12
13 : CTRL_R ( recopie en ligne courante la ligne memorisee )
14   LIGNE DUP DUP
15   ERA-LIGNE EDITOR R .LIGNE ; -->

```

```

SCR # 47
0 ( Editeur pleine page suite ABSEP86 )
1
2 : CTRL_I ( insere la ligne memorisee. la 15eme est perdue )
3   LIGNE EDITOR I
4   LIGNE 1+ 1 LOCATE 20 EMIT .ECRAN ;
5
6
7
8 : STOP ( --- : quitte l'edition )
9   2 MODE
10  [COMPILE] FORTH [COMPILE] DEFINITIONS QUIT ;
11
12
13 -->

```

```

SCR # 48
0 ( Editeur pleine page suite ABSEP86 )
1
2 : INIT ( theatre des prochaines scenes )
3   2 MODE
4   0 #WINDOW 25 3 80 5 WINDOW
5   EN-TETE
6   16 8 DO
7     FORTH I 4 + 1 LOCATE FORTH I 3 .R
8     FORTH I 4 + 70 LOCATE FORTH I .
9     LOOP
10  1 #WINDOW 23 6 73 9 WINDOW
11  CLS
12  TOP
13  .ECRAN CURSEUR ;
14 -->
15

```

```

SCR # 49
0 ( Editeur pleine page suite ABSEP86 )
1 : CHOIX
2   BEGIN KEY
3   CASE
4   243 ( fleche droite ) OF DROITE ENDOF
5   242 ( fleche gauche ) OF GAUCHE ENDOF
6   240 ( fleche haut ) OF HAUT ENDOF
7   241 ( fleche bas ) OF BAS ENDOF
8   248 ( ctrl fleche haut ) OF TOP ENDOF
9   249 ( ctrl fleche bas ) OF DEB-LIS ENDOF
10  250 ( ctrl fleche gauche ) OF DEB-L ENDOF
11  251 ( ctrl fleche droite ) OF FIN-L ENDOF
12  245 ( shift fleche bas ) OF SCR+1 ENDOF
13  244 ( shift fleche haut ) OF SCR-1 ENDOF
14  225 ( ctrl tab ) OF CTRL_TAB ENDOF
15  -->

```

```

SCR # 50
0 ( Editeur pleine page suite ABSEP86 )
1 24 ( ctrl X ) OF CTRL_X ENDOF
2 5 ( ctrl E ) OF CTRL_E ENDOF
3 4 ( ctrl D ) OF CTRL_D ENDOF
4 19 ( ctrl S ) OF CTRL_S ENDOF
5 8 ( ctrl H ) OF CTRL_H ENDOF
6 18 ( ctrl R ) OF CTRL_R ENDOF
7 9 ( ctrl I ) OF CTRL_I ENDOF
8 17 ( ctrl Q ) OF STOP ENDOF
9 224 ( COPY ) OF INS ENDOF
10 127 ( DEL ) OF DEL ENDOF
11 13 ( ENTER ) OF <ENTER> ENDOF
12 DUP ECRIT?
13   ENDCASE CURSEUR
14 AGAIN ;
15 -->

```

```

SCR # 51
0 ( Editeur pleine page fin ABSEP86 )
1 : EDIT
2   DECIMAL SCR ! INIT CHOIX ;
3
4 FORTH DEFINITIONS DECIMAL
5
6 : ED EDITOR EDIT ; FORTH
7
8 ;S
9
10 +-----+
11 | syntaxe: n ED ( n est l'ecran a editor ) |
12 +-----+
13

```



```

1
0 \ Chargement.
1
2
3 1 5 +THRU
4
5 55 VDD
6 CR .( Horloge chargée.)
7 CR .( Pour changer date/heure, utilisez DATE-HEURE.)
8 CR .( Pour avoir date/heure, utilisez NOW.)
9 48 VDD
10
11 6 VIEWS HORL.BLK \ place ce fichier dans VIEW-FILES.
12
13
14
15

```

```

2
0 \ Mois et Jours.
1
2 : "ARRAY (S lgr -- : compil. ; -- adr lgr : exécut. )
3 CREATE C, ASCII " WORD COUNT >R HERE R@ MOVE R> ALLOT
4 DOES> COUNT >R SWAP R@ + + R> ;
5
6 3 "ARRAY "MOIS "JanFevMarAvrMaiJunJuiAouSepOctNovDec"
7 3 "ARRAY "DAY "DimLunMarMerJeuVenSam"
8
9 CODE LIS-H (S fonct --- heure ou date empilée )
10 AX POP AL AH MOV 33 INT
11 DX PUSH \ sec. et 1/100 ou mois/jour
12 CX PUSH \ heure et min. ou année.
13 AL PUSH \ n° du jour de la semaine
14 NEXT C;
15

```

```

3
0 \ Horloge: CLK!, CLK@.
1
2 VARIABLE CLK! CLK! 7 ALLOT
3 : CLK@
4 42 LIS-H \ lecture de la date
5 256 /MOD DROP CLK! C! \ S;
6 SWAP 256 /MOD SWAP CLK! 1+ C! \ JJ
7 CLK! 2+ C! \ MM
8 CLK! 3+ ! \ AA
9 44 LIS-H \ lecture de l'heure
10 DROP CLK! 5+ 4 BOUNDS \ calcule limites de boucle
11 DO 256 /MOD I C! I 1+ C! 2 +LOOP \ place HHmmss en CLK!
12
13 : H? \ teste la présence de l'horloge.
14 CLK@ CLK! 1+ C@ 0= IF CR ." Horloge inactive."
15 FALSE ELSE TRUE THEN ;

```

```

4
0 \ Horloge: affichage.
1
2 : CLK# (S n --- )
3 CLK! + C@ S>D @ # ;
4 : (DATE) (S --- adr lgr )
5 CLK# <# 3 CLK! + @ 100 /MOD DROP S>D @S
6 2 CLK! + C@ 1- "MOIS DUP NEGATE HLD +! HLD @ SWAP CMOVE
7 1 CLK# #> ;
8 : (HEURE) (S --- adr lgr )
9 CLK# <# 7 CLK# ASCII : HOLD & CLK# ASCII : HOLD
10 5 CLK# #> ;
11 : JOUR CLK# CLK! C@ "DAY TYPE SPACE ;
12 : DATE (DATE) TYPE SPACE ;
13 : HEURE (HEURE) TYPE SPACE ;
14 : NOW H? IF 115 CURS 55 VDD 51 24 AT ." le "
15 JOUR DATE ." à " HEURE 48 VDD 117 CURS THEN ;

```

```

9
\ HORLOGE.
10Oct86JaD

Ce programme suppose la présence sur une carte multifonctions
de l'horloge "temps réel" du PC-XT, accessible par les fonctions
42 à 45 de l'interruption 21H.
fonct. 42 : lecture de la date, CX = année, DX = mois/jour.
      43 : positionne la date, mêmes valeurs.
      44 : lecture de l'heure, CX = hh/mm, DX = ss/100.
      45 : positionne l'heure, mêmes valeurs.

```

```

10
10Oct86JaD \ Mois et Jours.
22Oct86JaD

"ARRAY mot de définition pour les tableaux de chaînes.
"MOIS (S n --- ) n de 0 à 11
tableau des noms de mois.
"DAY (S n --- ) n de 0 à 6
tableau des noms de jour.
LIS-H lecture de la date (fonct=42) ou de l'heure (fonct=44).
Retourne les valeurs sur la pile.

```

```

11
22Oct86JaD \ Horloge: CLK!, CLK@.
22Oct86JaD

CLK! table contenant date/heure.
CLK@ lit la date et l'heure de l'horloge et les place
dans CLK!:
le 1er octet contient le n° du jour de la semaine
2, 3, 4 et 5èmes pour le jour, le mois et l'année;
6, 7, 8 et 9èmes pour heure, minute, sec. et 1/100.
H? teste la présence de l'horloge "temps réel", en supposant
qu'en son absence le contenu de l'octet est nul.

```

```

12
\ Horloge: affichage.
22Oct86JaD

CLK# convertit le n ième oct. de CLK! en ses codes ASCII.
(DATE) construit une chaîne contenant la date, format JJmmAA
(HEURE) construit une chaîne contenant l'heure, format HHmmss
JOUR affiche le jour de la semaine ( Lun, Mar, ... )
DATE " la date
HEURE " l'heure
NOW " date/heure en bas de l'écran.

```

5	13	220ct86JaD
0 \ EDITOR ID automatique.	100ct86JaD \ EDITOR ID automatique.	
1		
2 : (QUI) (S --) " JaD ;	(QUI) empile adr et lgr de la chaîne contenant l'identificateur de l'utilisateur.	
3		
4 : QUI (S --) (QUI) TYPE SPACE ;	QUI l'imprime.	
5		
6 : SET-ID (S --)	SET-ID remplace la routine usuelle de "boutage". Après	
7 H?	exécution de HELLO, et si l'horloge "temps réel" est	
8 IF (DATE) (EDITOR) ID SWAP CMOVE (QUI) ID 7 + SWAP CMOVE	présente, le tampon de l'éditeur ID contient la date	
9 THEN HELLO ;	et les initiales de l'utilisateur, qui seront placées	
10	par STAMP dans chaque écran modifié.	
11 ' SET-ID IS BOOT		
12		
13		
14		
15		
6	14	220ct82220ct86JaD
0 \ Mise à date/heure.	220ct86JaD \ Mise à date/heure.	
1		
2 CODE MIS-D (S JJ MM AAAa --- f1)	MIS-D les valeurs empilées JJ MM AAAa sont placées dans les	
3 CX POP AX POP AL DH MOV AX POP AL DL MOV 43 # AH MOV	registres de l'horloge.	
4 33 INT AH AH SUB AX PUSH NEXT C;		
5	MIS-H de mme pour l'heure HH mm (les secondes sont mises à 0)	
6 CODE MIS-H (S HH mm --- f1)		
7 AX POP AL CL MOV AX POP AL CH MOV 45 # AH MOV	Au retour, la pile contient un drapeau vrai (-1) si les	
8 DX DX SUB 33 INT AH AH SUB AX PUSH NEXT C;	valeurs de date ou d'heure ne sont pas valides.	
9		
10 : INPUT? (-- [n] f)	INPUT? entrée d'un nbr; empile seulement un drapeau faux s'il	
11 QUERY BL WORD NUMBER? NIP DUP 0= IF NIP THEN ;	y a eu entrée d'un autre caractère: la valeur concernée	
12 -->	n'est alors pas modifiée.	
13		
14		
15		
7	15	220ct86JaD
0 \ Mise à date/heure.	220ct86JaD \ Mise à date/heure.	
1		
2 : DATE-HEURE (S --)	DATE-HEURE demande de la date et de l'heure pour mise à jour.	
3 H? NOT IF EXIT THEN	Le jour JJ est un nbr compris entre 1 et 31.	
4 CR ." Date/heure courantes : " DATE HEURE CR	Le mois MM est un nbr compris entre 1 et 12.	
5 CR ." Jour ? " INPUT? IF 1 CLK! + C! THEN	L'année est comprise entre 1980 et ... 2047.	
6 CR ." Mois ? " INPUT? IF 2 CLK! + C! THEN	L'heure HH est comprise entre 0 et 23, les minutes	
7 CR ." Année ? " INPUT? IF 3 CLK! + C! THEN	mm entre 0 et 59; ne pas entrer de secondes.	
8 CR ." Heure ? " INPUT? IF 5 CLK! + C! THEN	Si un caractère autre qu'un chiffre est entré, la	
9 CR ." Minute ? " INPUT? IF 6 CLK! + C! THEN	valeur courante n'est pas modifiée.	
10 CR ." Frappez une touche pour prise en compte." KEY DROP CR	La prise en compte de ces valeurs se fait à l'appui	
11 CLK! 1+ C0 CLK! 2+ C0 3 CLK! + 0 MIS-D	d'une touche.	
12 IF ." Date non valide, recommencez." CR CR RECURSE THEN	Leur validité est vérifiée par le BIOS: en cas	
13 CLK! 5+ C0 CLK! 6+ C0 MIS-H	d'erreur, il faut recommencer.	
14 IF ." Heure non valide, recommencez." CR CR RECURSE THEN	Vérification à posteriori par NOW .	
15 NOW ;		

suite de la page 2

Ces quelques exemples très simples (et peu réalistes...) permettent, sans entrer dans les détails de la syntaxe, de donner une petite idée de la facilité et de la puissance que procure le langage APL dans la manipulation des variables multidimensionnelles. Pour faciliter la compréhension, nous avons utilisé des variables de faibles dimensions, mais les procédures utilisées sont indépendantes des dimensions des variables, lesquelles ne sont pratiquement limitées que par la mémoire disponible.

D'autre part, nous n'avons fait qu'effleurer le sujet, et l'APL possède bien d'autres possibilités de calcul.

Notamment, l'inversion d'une matrice qui est une affaire d'état avec d'autres langages, se programme en APL par un seul signe opératoire en forme de domino.

F.ESPINASSE

Bruce W. Walker, San Pedro, California
Forth Dimensions V/6
Traduction Eric Aubourg

Cet article donne une manière simple d'ajouter des structures de données au Forth, comme en PL/I, en COBOL ou en PASCAL. Il contient trois parties : Premièrement, ce que sont les structures de données et comment on peut les utiliser en Forth, deuxièmement comment fonctionnent les définitions Forth et troisièmement les définitions elles-mêmes.

Les structures de données.

Les programmes Forth utilisent en général uniquement deux types de données : les nombres et les tableaux de nombres. Alors que cela convient pour du calcul scientifique, les programmeurs de gestion savent depuis des temps immémoriaux (c.a.d. depuis 1960) que beaucoup de programmes sont plus faciles à concevoir et à écrire si l'on utilise le concept de structure de données. Une structure de données est une collection hétérogène de données, et on peut la concevoir soit comme une seule entité soit comme une collection avec des parties ayant des noms, selon la nécessité du moment.

Alors que le Forth a peu de possibilités de structures de données implémentées en standard, il a tous les outils nécessaires pour ajouter de nouveaux types de données. D'autres auteurs ont travaillé sur les piles utilisateur⁵, les nombres complexes², la quadruple précision⁴ et les chaînes de caractères^{3, 6}. La chose qui se rapproche le plus de structures de données en Forth est l'article sur la conception d'une base de données en Forth⁴; cet article est une mine d'or d'idées et tout programmeur Forth sérieux devrait le lire au moins une fois par an.

```
DECLARE 1 EMPLOYE,
      2 NOM CHARACTER (16),
      2 AGE FIXED BINARY (15,0);
```

Figure 1.

En PL/I, un programme peut avoir une déclaration comme celle de la figure 1. Cela signifie que EMPLOYE est composé de deux parties, NOM et AGE. On peut faire appel aux champs individuellement par EMPLOYE.NOM et EMPLOYE.AGE.

```
DECLARE 1 BOSS,
      2 NOM CHARACTER(16),
      2 AGE FIXED BINARY(15,0);
```

Figure 2.

L'avantage par rapport à deux variables séparées est que EMPLOYE peut être utilisé comme un tout. Par exemple, supposons que nous ayons aussi la déclaration de la figure deux. Alors l'instruction

BOSS = EMPLOYE
assigne les deux champs d'un coup.

```
18 SS.EMPLOYE
   S.MOVE EMPLOYE.MOVE
16 S.FLD NOM
  2 S.FLD AGE
```

Figure 3.

Pour faire la même chose en Forth, nous avons besoin de déclarer la structure, les champs, et créer des routines pour faire les assignements. Cela s'avère être très facile avec CREATE et DOES>. Les définitions sont données dans la section concluant cet article. L'exemple précédent de structure donnerait en Forth la figure 3. (Je n'ai pas inclu les fonctions de transfert, qui sont optionnelles.) Les références aux adresses des champs sont alors:

```
EMPLOYE NOM
EMPLOYE AGE
```

Et l'assignation en bloc est effectuée par:

```
EMPLOYE BOSS EMPLOYE.MOVE
```

(Notez que l'on ne redéclare pas nom et âge pour boss.)

Deux généralisations des structures de données apparaissent en PL/I. Premièrement, les éléments peuvent être des tableaux, comme dans la figure 4.

```
DECLARE 1 EMPLOYE(100),
      2 NOM CHARACTER(16),
      2 AGE FIXED BINARY(15,0);
```

Figure 4.

Ceci déclare un tableau de structures de données, ici 100 employés. Un programme peut alors contenir les instructions de la figure 5.

```
EMPLOYE(M) = EMPLOYE(N)
et
EMPLOYE(M).NOM = EMPLOYE(N).NOM
```

Figure 5.

Pour faire cela en Forth, nous créons un tableau d'éléments et utilisons la forme d'allocation dynamique de la déclaration de structure comme dans la figure 6.

```
1800 18 S.ARRAY EMPLOYE
      18 DS
      S.MOVE EMPLOYE.MOVE
16 S.FLD NOM
   FLD.MC NOM.MOVE
  2 S.FLD AGE
```

Figure 6.

```
N à EMPLOYE M à EMPLOYE
EMPLOYEE.MOVE
et
N à EMPLOYE M à EMPLOYEE NAME.MOVE
```

Figure 7.

La figure 7 est donc la version Forth de la figure 5.

La seconde généralisation est d'avoir plus d'un niveau, comme dans la figure huit. Maintenant une référence à un champ individuel ressemble à :

EMPLOYEE(N).NOM.INITIALES

```
DECLARE 1 EMPLOYEE(100),
  2 NOM,
  3 INITIALES CHARACTER(2),
  3 PATRONYME CHARACTER(14),
  2 AGE FIXED BINARY(15,0);
```

Figure 8.

En Forth, on peut faire la même chose en réinitialisant le décompte des adresses des champs, en déclarant la structure principale en premier, puis en déclarant les champs, comme dans le figure 9.

```
1800 18 S.ARRAY EMPLOYEE
  18 DS
  S.MOVE EMPLOYEE.MOVE
  16 S.FLD NOM
  FLD.MC NOM.MOVE
  2 S.FLD AGE
  0 NOM SF
  2 S.FLD INITIALES
  14 S.FLD PATRONYME
```

Figure 9.

Alors en Forth, la référence aux champs se fait par:

N à EMPLOYEE INITIALES

Une application typique d'une structure de données est une table avec deux champs : KY le champ clé, et VAL le champ valeur. Le tableau doit être maintenu trié selon le champ KY quand de nouvelles entrées sont ajoutées. Cela signifie que lorsque l'on ajoute une nouvelle entrée, une partie du travail concernera seulement le champ clé (c.a.d. le test pour trouver où la nouvelle entrée doit aller), tandis que l'autre partie concernera les deux champs (c.a.d. déplacer les autres entrées).

Au tableau de la figure dix est associé un compteur ACT qui donne le nombre actuel d'entrées utilisées. Le corps du tableau a deux champs, KY et VAL, tous les deux de 16 octets. £E est le nombre d'entrées maximum et a été défini comme une constante sur des bases générales de bonne programmation. On fait la convention que le tableau contient constamment une entrée bidon avec la plus grande valeur de VAL possible. Cela simplifie la recherche, puisque quand on ajoute une nouvelle entrée, on peut être sûr qu'on arrivera tôt ou tard sur un champ clé plus grand.

TBL.INIT initialise la table, et insère l'entrée bidon en première position, pour que la condition supposée par la routine de recherche soit vérifiée. TBL.F prend une entrée à ajouter sur la pile, et compare son champ clé avec les champs clés des entrées du tableau, jusqu'à ce qu'une entrée avec un champ clé plus grand soit trouvé. TBL.F laisse alors l'indice sur la pile. TBL.I prend un indice sur la pile et décale toutes les entrées suivantes d'un cran. Enfin, le mot TBL.INSERT supervise toute l'opération. Il vérifie si il reste de la place dans le tableau, et retourne 1 si ce n'est pas le cas. Sinon, il utilise TBL.F pour voir où l'entrée doit aller. Il utilise alors l'indice deux fois, une comme argument de TBL.I pour dégager de la place, et une autre comme argument de TBL.MOVE pour stocker la nouvelle valeur à l'endroit voulu. Finalement, il retourne 0 pour indiquer que l'entrée a bien été ajoutée au tableau.

```
1 VARIABLE ACT
25 CONSTANT £E
£E 4 * 4 S.ARRAY TBL
4 DS S.MOVE TBL.MOVE
2 S.FLD KY 2 S.FLD VAL

: TBL.F ACT à 0 DO
  DUP KY à I TBL KY à <
  IF DROP I LEAVE THEN LOOP ;

: TBL.I 1 - £E 1 - DO
  I TBL I 1 + TBL TBL.MOVE -1 + LOOP ;

: TBL.INSERT ACT à £E = IF DROP 1 ELSE
  DUP TBL.F DUP TBL.I TBL
  TBL.MOVE 0 THEN ;

: TBL.INIT 32767 0 TBLK KY ! 1 ACT ! ;
```

Figure 10.

Le fonctionnement des définitions.

Comme pour tous les mots CREATE ... DOES>, il y a une action au moment de la définition et une action au moment de l'exécution. Dans ce cas, l'action au moment de la définition comprend une sauvegarde des informations pour d'autres mots qui pourraient être utilisés plus tard, et un stockage dans le dictionnaire. Tous les mots utilisent trois variables pour sauvegarder des informations au moment de la définition:

- TLEN longueur totale de la structure
- CLEN longueur de la structure, jusqu'au champ sur lequel on travaille
- FLEN longueur du champ sur lequel on travaille

Ces variables sont utilisées pour calculer les données sauvegardées par les différentes routines de transfert (MOVE), donnée qui est utilisée par la partie (commune) DOES> des routines. L'exemple le plus simple est S.MOVE, qui transfère une structure entière. Quand une opération comme S.MOVE EMPLOYEE.MOVE est exécutée, le compilateur Forth utilise TLEN à , pour mettre la valeur courante de TLEN dans EMPLOYEE.MOVE. Quand

EMPLOYEE.MOVE est exécuté, l'adresse de EMPLOYEE.MOVE est empilée et la partie DOES > de S.MOVE est exécutée, et provoque un CMOVE qui transfère la structure entière.

Il y a juste quelques remarques à faire au sujet de l'utilisation des mots individuels. Il faut utiliser SS pour allouer une structure statique et commencer la définition des structures, seulement quand vous voulez à la fois les fonctions d'allocation et la structure de données. Quand la structure est déjà allouée, par un moyen ou un autre, il suffit d'utiliser DS pour commencer une structure dynamique. (La structure peut être déjà créée par exemple si elle a été définie par S.ARRAY, ou si on a créé un sous-champ, c'est à dire si on a relancé le décompte des adresses relatives).

Il peut sembler inutile d'avoir quatre routines pour transférer des champs (entre deux structures similaires, d'un champ à une adresse arbitraire, d'une adresse arbitraire vers un champ ou entre deux adresses arbitraires), mais on a besoin de toutes ces fonctions à un endroit ou à un autre; Je n'ai pas trouvé de noms plus astucieux pour les quatre cas. Les mots auraient pu être plus simples en passant des paramètres⁷, mais cela aurait ralenti l'exécution, et on risque d'appeler beaucoup les mots définissant les champs.

Description détaillée des mots.

Puisque la plupart des mots ont à la fois une action à la compilation et une action à l'exécution, chaque description a jusqu'à trois parties : Premièrement, l'entrée créée par ce mot dans le dictionnaire; deuxièmement l'action à la compilation; troisièmement l'action à l'exécution. Pour les mots plus complexes, il y a une description pas-à-pas du mot, avec des descriptions de la pile entre parenthèses.

INIT.SV (run time) Sauve la longueur totale de la structure dans TLEN, initialise le pointeur au début du champ courant, et stocke 0 dans CLEN.

STR.SV (run time) Sauve l'offset du champ et sa longueur; appelé par les mots de compilation et crée une entrée dictionnaire.

(entrée dictionnaire)

mot 1 = offset du champ

mot 2 = longueur du champ

SS (run time) alloue un tableau, puis appelle INIT.SV

DS (run time) synonyme de INIT.SV

S.ARRAY (compile time) Sauve la taille d'un élément, puis alloue un tableau:

(entrée dictionnaire)

mot 1 = taille d'un élément

mot 2 = nb d'éléments

(run time) multiplie la taille d'un élément par l'indice, et ajoute la base.

S.MOVE (compile time) Sauve la longueur totale :

(entrée dictionnaire)

mot 1 = longueur totale

(run time) Récupère la longueur qui avait été sauvée et utilise CMOVE (les adresses de départ et d'arrivée doivent déjà être sur la pile).

S.FLD (compile time) Sauve l'offset de début de ce champ, stocke la longueur du champ dans FLEN et l'ajoute au pointeur courant :

(entrée dictionnaire)

mot 1 = offset du champ.

(run time) Lit l'adresse relative du champ, et l'ajoute à l'adresse de base.

FLD.MA (compile time) Appelle STR.SV pour sauver les caractéristiques du champ :

(entrée dictionnaire)

mot 1 = offset du champ

mot 2 = longueur du champ

(run time) (adresse - adresse de la structure - pointeur dictionnaire)

DUP : Duplique l'adresse de l'entrée dans le dictionnaire.

2+ : Pointe vers la longueur du champ.

> R : Sauve l'adresse de la longueur du champ sur la pile de retour.

à : Lit l'offset du champ.

+ : Ajoute l'offset du champ à l'adresse de début de la structure.

R > : Récupère l'adresse de la longueur du champ.

à : Récupère la longueur du champ.

(adresse absolue du champ - longueur du champ)

CMOVE : transfère depuis l'adresse fournie sur la pile vers la structure de données.

FLD.MB (compile time) Appelle STR.SV pour sauver les caractéristiques du champ :

(entrée dictionnaire)

mot 1 = offset du champ

mot 2 = longueur du champ

(run time) (adresse de la structure - adresse - pointeur dictionnaire)

> R : Sauve le pointeur vers le dictionnaire.

SWAP : Echange les adresses des structures.

R : Récupère le pointeur vers le dictionnaire.

à : Lit l'offset du champ.

+ : Ajoute l'offset du champ à l'adresse de début de la structure.

R > : Récupère l'adresse de la longueur du champ.

à : Récupère la longueur du champ.

(adresse - adresse absolue du champ - longueur du champ) CMOVE : transfère depuis l'adresse fournie sur la pile vers la structure de données.

FLD.MC (compile time) Appelle STR.SV pour sauver les caractéristiques du champ :

(entrée dictionnaire)

mot 1 = offset du champ

mot 2 = longueur du champ

(run time) (adresse de la structure 1 - adresse de la structure 2 - pointeur dictionnaire)

> R : Sauve le pointeur vers le dictionnaire.

R : Récupère le pointeur vers le dictionnaire.

à : Lit l'offset du champ.

+ : Ajoute l'offset du champ à l'adresse de début de la structure 2.

(adresse de la structure 1 - adresse absolue du champ de la structure 2)

SWAP : Echange les adresses.

R : Récupère le pointeur vers le dictionnaire.

à : Lit l'offset du champ.

+ : Ajoute l'offset du champ à l'adresse de la structure 1.

SWAP : échange les adresses

R > : Prend le pointeur vers le dictionnaire.

2+ : Pointe vers la longueur du champ.

à : Lit la longueur du champ.

suite page 14

```

1
0 \ Mots: CLS HOME CASE OF ENDOF ENDCASE
1 : CLS 12 EMIT BLINK OFF BOUT OFF ;
2 : CLS IS DARK
3 : HOME 30 EMIT BLINK OFF BOUT OFF ;
4
5 : CASE CSP @ !CSP ; IMMEDIATE
6
7 : OF COMPIL OVER COMPIL = COMPIL ?BRANCH >MARK
8 COMPIL DROP ; IMMEDIATE
9 : ENDOF COMPIL BRANCH >MARK SWAP
10 >RESOLVE ; IMMEDIATE
11 : ENDCASE COMPIL DROP
12 BEGIN SP@ CSP @ <
13 WHILE >RESOLVE
14 REPEAT
15 CSP ! ; IMMEDIATE
    
```

```

2
0 \ Mots: PLOT PLOT DRAW DRAW
1 HEX
2 CODE PLOT
3 D1 C, E1 C, C5 C, CD C, BE9B ,
4 BBEA , C1 C, NEXT END-CODE
5 CODE PLOT
6 D1 C, E1 C, C5 C, CD C, BE9B ,
7 BBED , C1 C, NEXT END-CODE
8 CODE DRAW
9 D1 C, E1 C, C5 C, CD C, BE9B ,
10 BBF6 , C1 C, NEXT END-CODE
11 CODE DRAW
12 D1 C, E1 C, C5 C, CD C, BE9B ,
13 BBF9 , C1 C, NEXT END-CODE
14 DECIMAL
15
    
```

```

3
0 \ Mots: TAG TRANS #WINDOW WINDOW
1 HEX
2 CODE TAG
3 E1 C, C5 C, 7D C, CD C, BE9B ,
4 BB63 , C1 C, NEXT END-CODE
5 : TRANS
6 22 EMIT EMIT ;
7 CODE #WINDOW
8 E1 C, C5 C, 7D C, CD C, BE9B ,
9 BB84 , C1 C, NEXT END-CODE
10 DECIMAL
11 : WINDOW
12 26 EMIT 1- EMIT 1- EMIT 1- EMIT 1- EMIT ;
13
14
15
    
```

```

4
0 \ Mots: MODE PAPER PEN INVERSE INK BORDER LOCATE
1 : MODE
2 4 EMIT EMIT EMIT ;
3 : PAPER
4 14 EMIT EMIT ;
5 : PEN
6 15 EMIT EMIT ;
7 : INVERSE
8 24 EMIT ;
9 : INK
10 28 EMIT EMIT EMIT EMIT ;
11 : BORDER
12 29 EMIT EMIT EMIT ;
13 : LOCATE
14 31 EMIT EMIT EMIT ;
15 \ LOCATE IS AT ( En option)
    
```

```

11
09dec86 MP Commentaires de CLS HOME CASE OF ENDOF ENDCASE 09dec86 MP
CLS ( ---) efface le contenu de l'affichage video
sur AMSTRAD/SCHNEIDER CPC 464, 664 et 6128
HOME ( ---) ramene le curseur en ligne 0, colonne 0
sur AMSTRAD/SCHNEIDER CPC 464, 664 et 6128.
    
```

CASE OF ENDOF ENDCASE sont des mots gerant une structure de controle, dont voici un exemple:

```

: jour ( n ---)
case 1 of ." Lundi " endof
2 of ." Mardi " endof
3 of ." Mercredi " endof
...etc...
7 of ." Dimanche " endof
endcase ; et '4 jour' affiche 'Jeudi'
    
```

```

12
Commentaires de PLOT PLOT DRAW DRAW 09dec86 MP
PLOT ( x y ---) Affiche un point graphique aux coordonnees x y
dans la couleur selectionnee par GPEN.
    
```

PLOT (dx dy ---) Affiche un point graphique aux coordonnees dx dy relatives au trace du precedent point

DRAW (x y ---) Trace un trait graphique depuis le dernier point affiche vers le point de coordonnees x y.

DRAW (dx dy ---) Trace un trait graphique depuis le dernier point affiche vers le point de coordonnees relatives dx dy

```

13
09dec86 MP Commentaires de TAG TRANS #WINDOW WINDOW 09
TAG ( u ---) Selectionne mode ecriture texte en coordonnees
graphiques: u=1 selectionne, u=0 deselectionne
    
```

TRANS (u ---) Selectionne mode trace transparent graphique et texte: u=1 selectionne, u=0 deselectionne

#WINDOW (u ---) Selectionne le numero de fenetre courant u compris dans l'intervalle 0..7

WINDOW (u1 u2 u3 u4 ---) Definit la taille de la fenetre courante. Exemple:

```
3 #WINDOW 25 20 60 30 WINDOW
```

```

14
09dec86 MP Commentaires de MODE PAPER PEN INVERSE INK BORDER LOCATE 09dec86 MP
MODE ( u ---) Selection du mode d'affichage:
u=0 20 car/ u=1 40 car/ u=2 80 car/ligne
PAPER ( u ---) selection couleur fond en mode texte
u compris dans l'intervalle 0..26
PEN ( u ---) selection couleur forme en mode texte
u compris dans l'intervalle 0..26
INVERSE ( ---) Inversion des couleurs fond et encre en mode
texte.
INK ( u1 u2 u3 ---) Etablit couleurs u1 u2 de l'encre u3:
u1 u2 compris 0..26, u3 compris 0..15
BORDER ( u1 u2 u3 ---) Etablit les couleurs u1 u2 du bord de
    
```

FORTH 83-Standard sous CP/M pour AMSTRAD CPC 464/664/6128

5
 0 \ Mots: FLASH ?CHAR CLG ORIGIN
 1 HEX
 2 CODE FLASH
 3 E1 C, D1 C, C5 C, 63 C, CD C, BE98 ,
 4 BC3E , C1 C, NEXT END-CODE
 5 CODE ?CHAR
 6 C5 C, CD C, BE98 ,
 7 B860 , 26 C, 00 C, 6F C, C1 C, E5 C, NEXT END-CODE
 8 CODE CLG
 9 C5 C, CD C, BE98 ,
 10 B808 , C1 C, NEXT END-CODE
 11 CODE ORIGIN
 12 D1 C, E1 C, C5 C, CD C, BE98 ,
 13 B8C9 , C1 C, NEXT END-CODE
 14 DECIMAL
 15

6
 0 \ Mots: GPEN GPAPER GWINDOW VSWAP
 1 HEX
 2 CODE GPEN
 3 E1 C, C5 C, 7D C, CD C, BE98 ,
 4 B8DE , C1 C, NEXT END-CODE
 5 CODE GPAPER
 6 E1 C, C5 C, 7D C, CD C, BE98 ,
 7 B8E4 , C1 C, NEXT END-CODE
 8 CODE GWINDOW
 9 E1 C, D1 C, C5 C, CD C, BE98 ,
 10 B8CF , C1 C, E1 C, D1 C, C5 C, CD C, BE98 ,
 11 B8D2 , C1 C, C3 NEXT END-CODE
 12 CODE VSWAP
 13 E1 C, D1 C, C5 C, 45 C, 48 C, CD C, BE98 ,
 14 B8B7 , C1 C, NEXT END-CODE
 15 DECIMAL

7
 0 \ Mots: MOVE MOVER
 1 HEX
 2 CODE MOVE
 3 D1 C, E1 C, C5 C, CD C, BE98 ,
 4 B8C0 , C1 C, NEXT END-CODE
 5 CODE MOVER
 6 D1 C, E1 C, C5 C, CD C, BE98 ,
 7 B8C3 , C1 C, NEXT END-CODE
 8 DECIMAL
 9
 10
 11
 12
 13
 14
 15

8
 0 \ Table des sinus 16 bits
 1 HERE
 2 0000 , 0175 , 0349 , 0523 , 0698 , 0872 , 1045 , 1219 ,
 3 1392 , 1564 , 1736 , 1908 , 2079 , 2250 , 2419 , 2588 ,
 4 2756 , 2924 , 3090 , 3256 , 3420 , 3584 , 3746 , 3907 ,
 5 4067 , 4226 , 4384 , 4540 , 4695 , 4848 , 5000 , 5150 ,
 6 5299 , 5446 , 5592 , 5736 , 5878 , 6018 , 6157 , 6293 ,
 7 6428 , 6561 , 6691 , 6820 , 6947 , 7071 , 7193 , 7314 ,
 8 7431 , 7547 , 7660 , 7771 , 7880 , 7986 , 8090 , 8192 ,
 9 8290 , 8387 , 8480 , 8572 , 8660 , 8746 , 8829 , 8910 ,
 10 8988 , 9063 , 9135 , 9205 , 9272 , 9336 , 9397 , 9455 ,
 11 9511 , 9563 , 9613 , 9659 , 9703 , 9744 , 9781 , 9816 ,
 12 9848 , 9877 , 9903 , 9925 , 9945 , 9962 , 9976 , 9986 ,
 13 9994 , 9998 , 10000 ,
 14 : SINNEG DUP 90 > IF 180 SWAP - THEN [0 SWAP] LITERAL
 15 SWAP 2 * + 0 ; DROP

15
 09dec86 MP \ Mots: FLASH ?CHAR CLG ORIGIN 09dec86 MP
 FLASH (u1 u2 ---) Etablit la duree des clignotements. u1 et u2
 sont donnees en 1/50e de seconde.
 ?CHAR (--- c) Delivre le code du caractere situe a l'empla-
 cement du curseur. Exemple: 10 15 LOCATE ?CHAR
 CLG (---) Vide le contenu de la fenetre graphique.
 ORIGIN (y x ---) Precise l'origine de la fenetre graphique.
 Soit 0<=y<=399 et 0<=x<=639, exemple:
 399 200 639 300 GWINDOW 200 300 ORIGIN

16
 09dec86 MP \ Mots: GPEN GPAPER GWINDOW VSWAP 09dec86 MP
 GPEN (u ---) Etablit la couleur u du curseur graphique.
 Soit 0<=u<=255
 GPAPER (u ---) Etablit la couleur de fond en mode graphique.
 Soit 0<=u<=255.
 GWINDOW (u1 u2 u3 u4 ---) Determine les coordonnees de la fe-
 netre graphique. u1=y haut, u2=y bas,
 u3=x droite, u4=x gauche.
 VSWAP (u1 u2 ---) Echange les caracteristiques des fenetres
 u1 et u2.

17
 \ Mots: MOVE MOVER
 MOVE (y x ---) Deplace le curseur aux coordonnees y x.
 MOVER (y x ---) Deplace le curseur aux coordonnees
 relatives y x.

18
 \ Table des sinus 16 bits
 Cette table a ete composee a partir de la formule:
 n=int(10000*sin(a))
 pour 0<=a<=90

SINNEG Delivre le sinus signe.

```

9
0 \ Fonctions trigonometriques
1 : $IN
2 360 MOD DUP 0< IF 360 + THEN
3 DUP 180 > IF 180 - SINNEG -1 * ELSE SINNEG THEN ;
4 : $SIN
5 SIN 10000 1/MOD SWAP 5000 > IF 1 + THEN ;
6 : $COS
7 90 + $SIN ;
8 VARIABLE X0 VARIABLE Y0 VARIABLE DEB VARIABLE FIN
9 : CENTER
10 Y0 ! X0 ! ;
11 : POSXIV1
12 OVER OVER $COS X0 0 +
13 ROT ROT $SIN Y0 0 + ;
14
15

```

```

19
\ Fonctions trigonometriques
SIN    delivre le sin(a modulo 360)

$SIN   ( n deg --- sin) Delivre le sinus d'un angle multiplie
        par n. Exemple: 100 60 $SIN . affiche 50
$COS   idem a $SIN mais applique au cosinus.

X0 Y0 DEB FIN variables utilisees pour le trace d'arc de cercle
CENTER ( x y ---) Affecte le centre d'un arc de cercle.

POSXIV1 Calcule la position d'un point par rapport au centre
        lors d'un trace d'arc de cercle.

```

suite de la page 11

CMOVE : Transfert.
FLD.MD (compile time) Sauve la longueur du champ.
 (entrée dictionnaire)
 mot 1 = longueur du champ
 (run time) Lit la longueur du champ et effectue
 le transfert entre les deux adresses.

Bibliographie

1. Beers, David A. "Quadruple Word Simple Arithmetic" *Forth Dimensions IV/1*.
2. Clark, Alfred, "Complex Analysis in Forth" *Forth Dimensions III/4*.
3. Harris, Kim, "Forth Extensibility" *BYTE, Vol 5 No 8 (Aout 1980)*.
4. Haydon, Glen B., "Elements of Forth Data Base Design", *Forth Dimensions III/2*.
5. Helmers, Peter H. "Userstack" *Forth Dimensions III/1*.
6. McCourt, Michael and Marisa, Richard A., "The String Stack", *Forth Dimensions III/4*.
7. McKibbin, David, "Parameter Passing to DOES >", *Forth Dimensions III/1*.



ICI & MAINTENANT!

Tous les mercredis, de 18h à 20h, des sujets variés, un niveau élevé, des invités prestigieux, des idées nouvelles et audacieuses, le tout animé par Michel ROUSSEAU. ICI et MAINTENANT est accessible également en composant le 3615 code X20001.

```

ECRAN No 1 FENETRE
FORGET ME : ME ; DECIMAL
0 VARIABLE Curs 0 , 4 CONSTANT 4
10 CONSTANT D_OC 20 CONSTANT N_OC
50 CONSTANT D_Y 100 CONSTANT N_Y
: Cadre 2OVER D+ OVER 4 PICK 2OVER
LINE 2OVER TO 4 ROLL OVER TO TO DROP ;
: CURS! 65217 VC6 65218 VC6 curs 2! ;
( sauvegarde position du curseur )
: CURS6 curs 26 SWAP AT ; CURS!
: >Pim D_OC D_Y N_OC N_Y BPUSH ;
: twin D_OC 4 * 4 + D_Y N_Y + 4 -
N_OC 2- 4 * N_Y 8 - WINDOW ;
: dims D_OC 4 * 2+ D_Y 2+
N_OC 4 * 4 - N_Y 4 - ;
: >cst ' N_Y ! ' N_OC !
' D_Y ! ' D_OC ! ;
: Fen STANDARD 0 BRIGHT
1 EMIT >cst >Pim twin
2 INK dims 2OVER D+ BOX
3 INK dims cadre
PAPER PAGE PEN ; -->

```

```

ECRAN No 2 Echange d'écrans
: Sortie BPOP 0 PAPER 2 PEN
1 BRIGHT 1 EMIT CURS6 ;
: 1ECR 3 1 20 20 32 40 Fen ;
: 2ECR 3 2 20 100 32 80 Fen ;
: 3ECR 2 1 32 50 26 100 Fen ;
( nom Pen Paper octet/X/origine
Pixel/Y/origine octet/X/long
Pixel/Y/haut -- Fen )

```

```

0 VARIABLE STOCK 7296 ALLOT
7296 CONSTANT 1/2e ( 1/2 écran )
49152 CONSTANT Video ( départ vidéo )
34048 CONSTANT Ch/II ( CHROMA II )
: -><- ( échange de Page )
Video STOCK 1/2e VCMOVE
Ch/II video 1/2e VCMOVE
STOCK Ch/II 1/2e VCMOVE
Video 1/2e + STOCK 1/2e VCMOVE
Ch/II 1/2e + Video 1/2e + 1/2e VCMOVE
STOCK Ch/II 1/2e + 1/2e VCMOVE
Video STOCK 1/2e VCMOVE ; -->

```

```

ECRAN No 3 VOIR
: >SAUV
Video 1/2e + STOCK 1/2e VCMOVE
STOCK
[ Ch/II 1/2e + 1/2e + 64 + ] LITERAL
1/2e CMOVE
Video STOCK 1/2e VCMOVE ;
: SAUV>
STOCK Video 1/2e VCMOVE
[ Ch/II 1/2e + 1/2e + 64 + ] LITERAL
STOCK 1/2e CMOVE
STOCK [ Video 1/2e + ] LITERAL
1/2e VCMOVE ;
: R/IN sortie >SAUV CLS 1ECR ;
: R/OUT sortie CLS SAUV> 1ECR ;
( R/IN sauve l'écran dans STOCK et en
fin de mémoire , R/OUT l'affiche )

```

```

: VOIR BPOP -><- 1ECR ."■■ VOIR"
CR ."■■ ++++" QUIT ;
BINIT 1ECR ."■■ VOIR"
CR ."■■ ----" CR -->

```

```

ECRAN No 4 DBLCOP
0 VARIABLE #A
: ↑↑ I' 0= IF 1 ELSE I' 1 =
IF 2 ELSE I' 0 DO 2 LOOP
I' 1- 0 DO * LOOP THEN THEN ;
: DBLCOP FAST sortie
CR CR 0 #A ! 8 LEMIT
1 227 DO I
241 1 DO 7 0 DO J OVER I - POINT 0=
IF ↑↑ #A +! THEN LOOP
#A à 128 + LEMIT 0 #A !
LOOP -><-
241 1 DO 7 0 DO J OVER I - POINT 0=
IF ↑↑ #A +! THEN LOOP
#A à 128 + LEMIT 0 #A !
LOOP -><-
DROP 13 LEMIT
GET 13 = IF LEAVE THEN
-7 +LOOP
15 LEMIT 13 LEMIT 1ECR SLOW ; -->

```

```

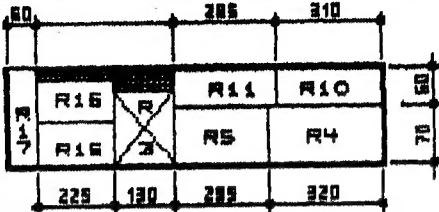
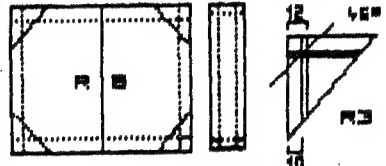
ECRAN No 5 Explications
( HECTOR HRX ----- Jean Michel
ANZIL
Fenêtrage à la demande avec sauve-
garde de la position du curseur et
sauvegarde de la portion d'écran dans
la Pile image

```

"sortie" = retour écran .
 DBLCOP permet d'imprimer deux écrans
 sur toute la largeur de l'imprimante
 SEIKOSHA GP 500A en mode graphique
 480 Points . Cette version est adap-
 tée au logiciel graphique CHROMA II
 écrit en BASIC adresse dessin 34048
 Fonctionnement: 1er écran chargé par
 RESET/INIT et l'option 2 du menu HRX
 "K7" . Tapez VOIR et sauvez par R/IN
 Chargez le 2eme par INIT et option 2
 Tapez R/OUT pour revoir le 1er
 tapez VOIR pour vérifier que tout le
 monde est là . Puis DBLCOP pour impri-
 mer les deux pages . Nov*1986)

HECTOR HRX

Exemple d'exécution de DBLCOP

PORTEFEUILLE A ABATTANTS		#A/N1	COLLEGE	CLASSE
NOM.		PRENOM.		
OPERATIONS	CROQUIS	OBSERVATIONS	OUTILS	
DERIT		ATTENTION R4=R5= 70MM R10=R11=R15= 50MM		
CARTON 2MM R1 R2				
CARTON 1MM R7R8 R13 R14				
TOILE ➡				
COUVRURE R5 450 x 360 mm				
TOILE 1030 X 130 mm				
TRACAGE				
TRACER SUR R5 4 RESERVES DE 20mm 4 TRIANGLES RECT. 150x150 DE 80mm DE COTE				

Opérations en notation scientifique

par G. SOULA

0 (DIVISION NOTATION SCIENTIFIQUE)

1

2 2VARIABLE N1 VARIABLE N2 VARIABLE DIV VARIABLE EXP

3 VARIABLE P1 VARIABLE P2 VARIABLE QUOT 10 ALLOT

4

5 : F1 2INPUT N1 2! ;

6 : F2 INPUT DUP 32766 UK

7 IF N2 !

8 ELSE . ." TROP GRAND! " THEN ;

9

10 : QUOTIENTS

11 DUP DIV ! 5 0

12 DO U/ I 2* QUOT + !

13 10 U* DIV @

14 LOOP DROP 2DROP ;

15

0 (DIVISION NOTATION SCIENTIFIQUE)

1

2 : 4DECIM 5 0

3 DO QUOT I 2* + @

4 I DUP 0=

5 IF DROP 10000 U*

6 ELSE DUP 1 =

7 IF DROP 1000 U*

8 ELSE DUP 2 =

9 IF DROP 100 U*

10 ELSE DUP 3 =

11 IF DROP 10 U*

12 ELSE DUP 4 =

13 IF DROP 1 U* THEN

14 THEN THEN THEN THEN

15 LOOP D+ D+ D+ D+ ;

```

0 ( DIVISION NOTATION SCIENTIFIQUE )
1
2 : E1 ." E? " INPUT P1 ! ;      : E2 ." E? " INPUT P2 ! ;
3 : 1ER-NB ." 1ERNB? " F1 E1 ;
4 : 2EME-NB ." 2EME NB? " F2 E2 ;
5
6 : ENTREE CR 1ER-NB CR 2EME-NB ;
7
8 : CALCULS
9 ENTREE N1 2@ N2 @ QUOTIENTS
10 4DECIM P1 @ P2 @ - EXP ! ;
11
12 : AFFICHE
13 CR <# 69 HOLD # # # # 46 HOLD #S #> TYPE EXP @ . ;
14
15 : F/ CALCULS AFFICHE ;

```

```

0 ( MULTIPLICATION NOTATION SCIENTIFIQUE )
1
2 VARIABLE COMPTER VARIABLE EXP
3 VARIABLE N1 20 ALLOT VARIABLE N2 20 ALLOT
4
5 : F1
6 ." 1ER NB? " N1 12 EXPECT 0 EXP !
7 12 0 DO N1 1 + C@ 0= IF LEAVE 1 COMPTER ! THEN LOOP
8 N1 1- NUMBER
9 COMPTER @ ;
10
11 : F2
12 ." 2EME NB? " N2 12 EXPECT
13 12 0 DO N2 1 + C@ 0= IF LEAVE 1 COMPTER ! THEN LOOP
14 N2 1- NUMBER
15 COMPTER @ ;

```

```

0 ( MULTIPLICATION NOTATION SCIENTIFIQUE )
1
2 : OPAD PAD 20 ERASE ;
3
4 : REDUIT
5 DUP 9 >
6 IF ." NOMBRE TROP GRAND "
7 DROP 2DROP QUIT
8 ELSE DUP 4 > IF 4 - DUP EXP +!
9 0 DO # LOOP
10 ELSE DROP 0 EXP +!
11 THEN
12 OPAD ;
13
14
15

```

```

0 ( MULTIPLICATION NOTATION SCIENTIFIQUE )
1
2 : F*
3 CR F1 REDUIT
4 CR F2 REDUIT
5 DD* 10000 U/
6 SWAP 5 > IF 1+ THEN 0 ;
7
8 : AFFICHE <# 69 HOLD #S #> TYPE EXP @ 4 + . ;
9
10 : F*. F* AFFICHE ;
11
12
13
14
15

```

```

0 ( NOTATION SCIENTIFIQUE )
1
2 : ?DPL
3 DPL @
4 DUP 0<
5 IF DROP 0
6 ELSE DROP -1
7 THEN ;
8
9 : CC
10 ?DPL 0
11 BEGIN 1+ DUP
12 TIB @ + C@ DUP 32 =
13 IF DROP 1
14 ELSE DROP 0 THEN
15 UNTIL ;

```

```

0 ( NOTATION SCIENTIFIQUE )
1
2 VARIABLE EXP
3
4 : AFFICHE
5 <# 69 HOLD #S #>
6 TYPE
7 EXP ? ;
8
9
10 1 SWAP ?DUP
11 IF 0 DO OVER # LOOP
12 THEN
13 SWAP DROP ;
14
15

```

```

0 ( NOTATION SCIENTIFIQUE )
1
2 : (F)
3 0 EXP !
4 CC DUP 9 >
5 IF ." NOMBRE TROP ELEVE "
6 DROP 2DROP
7 ELSE DUP 5>
8 IF 5 - DUP EXP !
9 0 DO # LOOP AFFICHE
10 ELSE DROP 0 EXP !
11 AFFICHE
12 THEN
13 THEN ;
14
15

```

La première chose à faire est de créer un fichier, en précisant sa taille en blocs:

CREATE-FILE <nom-de-fichiers> (n ---)
crée un fichier avec n blocs.

Dès lors, vous pourrez ouvrir le fichier:

OPEN <nom-de-fichier>

Mais OPEN fait bien mieux: il crée, si cela n'a pas été fait, le mot <nom-de-fichier> qui, à l'exécution, ouvrira le fichier. Exemple:

```
10 CREATE-FILE FICHIER1 crée FICHIER1
avec 10 blocs (0 à 9)
OPEN FICHIER1 ouvre FICHIER1 et
fabrique le mot FICHIER1
FICHIER1 réouvre FICHIER1
```

Le mot MORE (n ---) ajoute n blocs au fichier existant. Attention, certaines versions "oublient" de sauvegarder la nouvelle configuration du fichier! Le remède est de redéfinir MORE comme suit:

```
: MORE CAPACITY SWAP 8* MAXREC= +! CAPACITY
DO I BLOCK B/BUF BLANK UPDATE LOOP FLUSH ;
```

La copie de blocs est exécutée à l'aide du mot COPY (b1 b2 ---), ce qui a pour action de copier le contenu du bloc b1 dans le bloc b2.

La copie de blocs multiples est exécutée par CONVEY et utilisée comme suit:

b1 b2 TO b3 CONVEY

Exemple:

10 20 TO 15 CONVEY

copie les blocs 10 à 20 dans les blocs 15 à 25. Selon le cas, CONVEY fera les copies dans l'ordre croissant ou décroissant, ceci pour éviter les chevauchements de blocs à copier. Vous n'avez donc pas à vous en préoccuper.

Le standard F83 prévoit la gestion simultanée de deux fichiers, le fichier courant et le fichier secondaire (IN-FILE) que nous appellerons fichier donneur. Les concepteurs du F83 ont prévu, dans le vocabulaire FILES, de nouvelles versions des mots LOAD COPY et CONVEY effectuant respectivement le chargement, la copie simple et la copie multiple à partir du fichier secondaire. Efficace, mais dangereux pour les distraits oubliant quel est le fichier primaire et quel est le fichier secondaire!

Un mot permet d'ouvrir le fichier secondaire: FROM. Il s'utilise sous la forme:

FROM <nom-de-fichier>

Ce mot donne également accès au vocabulaire FILES et donc aux versions spéciales de LOAD COPY et CONVEY. Dans les versions 1xx du F83, ces mots ramènent au vocabulaire initial après exécution. Ils ne servent donc qu'une fois dans les versions suivantes, ils restent actifs tant que vous n'avez pas explicitement quitté le vocabulaire FILES. Exemple, première version de F83:

```
FORTH
OPEN FICHIER1 ouvre FICHIER1
FROM FICHIER2 ouvre FICHIER2 et
pointe sur le vocabulaire FILES
1 1 COPY copie le bloc 1 de fichier 2
dans le bloc 1 de FICHIER1 et ramène
au vocabulaire FORTH.
```

3 4 COPY copie le bloc 3 de FICHIER1
dans le bloc 4 de FICHIER1.

Exemple pour les versions suivantes:

```
FORTH
OPEN FICHIER1 ouvre FICHIER1
FROM FICHIER2 ouvre FICHIER2 et
pointe sur le vocabulaire FILES
1 1 COPY copie le bloc 1 de fichier 2
dans le bloc 1 de FICHIER1
3 4 COPY copie le bloc 3 de FICHIER2
dans le bloc 4 de FICHIER1.
vous êtes toujours dans le vocabulaire FILES
```

Le mot SWITCH permute le fichier courant et le fichier donneur. Si vous n'en disposez pas, définissez-le comme suit:

```
: SWITCH
FILE @ IN-FILE @ FILE ! IN-FILE ! ;
```

Voici, pour votre tranquillité d'esprit, un mot qui ramènera les deux fichiers au seul fichier courant:

```
: UNIQUE FILE @ IN-FILE ! ;
```

Note: IN-FILE est une variable contenant le FCB du fichier donneur. Dans certaines versions, cette variable s'appelle >FROM. Remplacez donc, si besoin est, IN-FILE par >FROM dans les deux définitions ci-dessus.

MANIPULATIONS POUR SAUVEGARDE DES BLOCS SUR AMSTRAD PCW

par Marc PETREMAN

C'est bien joli, le FORTH 83-Standard est compatible en version CP/M sur toute la gamme AMSTRAD, mais c'était trop beau pour durer. Mr BRUN Nicolas a trouvé l'os; le mot DONE exécuté en fin d'édition ne sauvegarde rien.

C'est que Mr AMSTRAD, ne voulant rien faire comme tout le monde (il était très bien pourtant le CP/M 2.0) a imaginé d'embêter le peuple avec un CP/M 3.0 appelé aussi CP/M+. Et ce CP/M+ a comme principale propriété de ne pas permettre la compatibilité avec les logiciels tournant sous CP/M 2.0, certains du moins...

En effet, le PCW8256 ou PCW8512 se sert de son extension mémoire comme d'un disque virtuel. Nous disposons donc sous CP/M+ de deux drives voir trois:

```
1 drive physique: A: M:
2 drives physiques: A: B: M:
```

Or, quand vous éditez un programme sous FORTH, l'option W utilisée sous EDITOR ou SAVE-BUFFERS écrit le contenu de vos blocs sur M: et non sur A: ou B:. Donc, s'il vous prend l'envie de quitter FORTH puis d'y revenir, vous aurez la désagréable surprise de constater que rien n'a été sauvegardé. La solution à ce remède est simple, il suffit de taper:

```
DONE CP/M CLOSE FORTH
```

ou de définir:

```
ONLY CP/M ALSO FORTH ALSO
: TERMINE DONE CLOSE ;
ONLY FORTH ALSO
```

et d'utiliser TERMINE à la place de DONE. Cette définition n'est pas incompatible avec F83 sur AMSTRAD CPC464, CPC664 ou CPC6128. Sur IBM et compatibles, on remplacera CP/M par DOS.